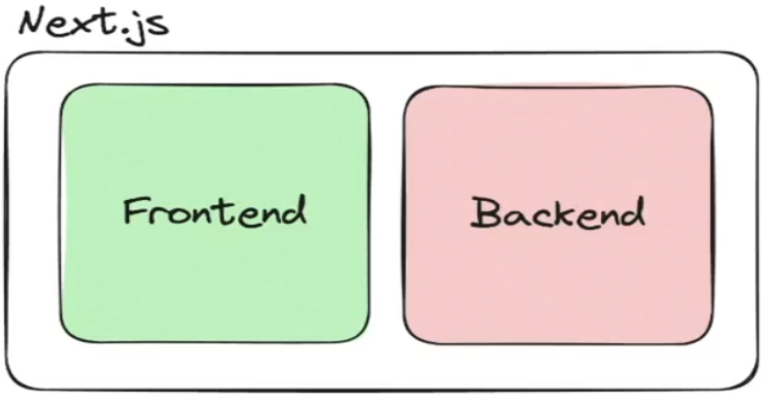
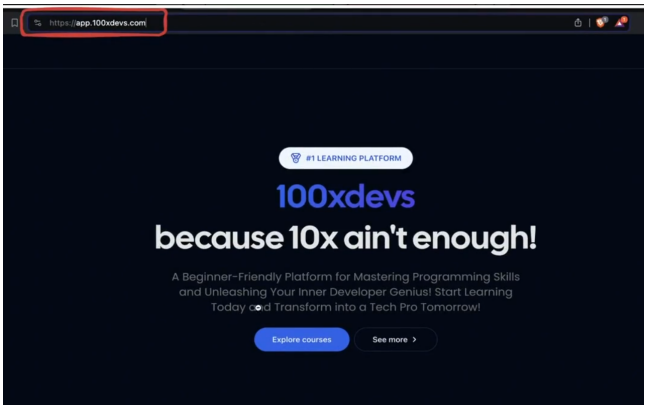
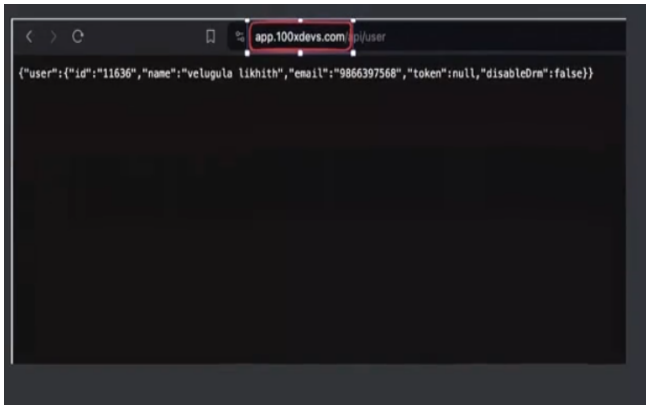
Next Js for Backend (Server side code)

`Next.js` is a full stack framework



This simply means that you dont really need `express` as well as this simply means that same `process` can handle both frontend and backend code.





The left pic consists of the `HTML` code being returned so `frontend`

The `json` which you see in the right part of the pic is generally returned by the `backend` server.

Benefits ->

- **Single codebase for all your codebase**
- **No `CORS` issue, single domain name for your FE and BE**

- **Ease of deployment, deploy a single codebase** (will see later) [basically your frontend if written in `react`, then will be deployed on `S3`(**Object Store**) and backend if written in `node.js` then it will be deployed on `EC2` but if using `next.js`, then both frontend and backend will be deployed on the same thing]

Recap of Data fetching in `React`

Lets do a quick recap of how data fetching works in `React`.

We're not building backend yet. Assume you already have this backend route -> https://week-13-offline.kirattechnologies.workers.dev/api/v1/user/details

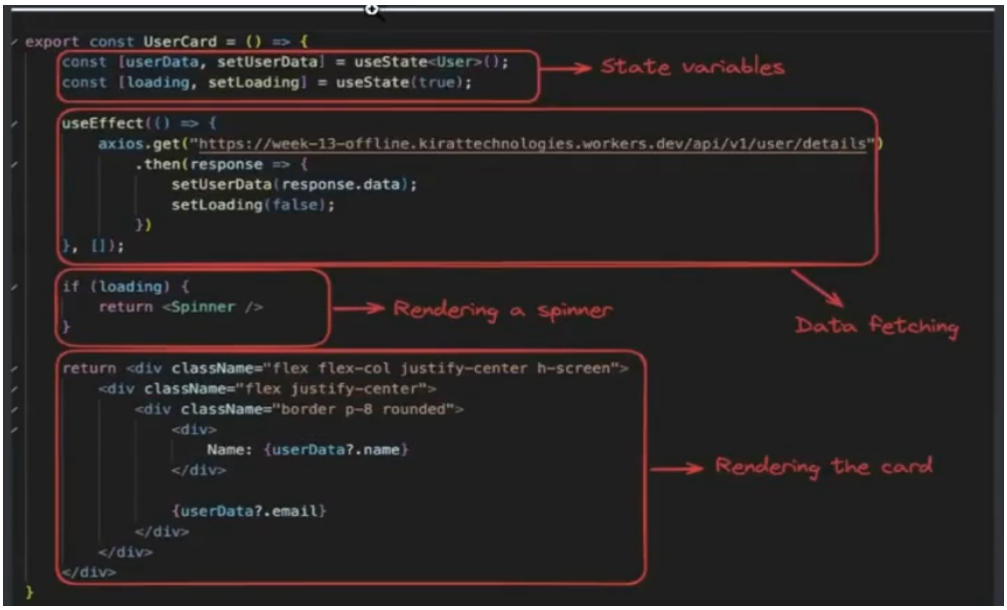
code :- https://github.com/100xdevs-cohort-2/week-14-2.1

Website :- https://week-14-2-1.vercel.app/

User card website

Build a website that lets user see their name and email from the given endpoint

The UserCard component made above looks something like this



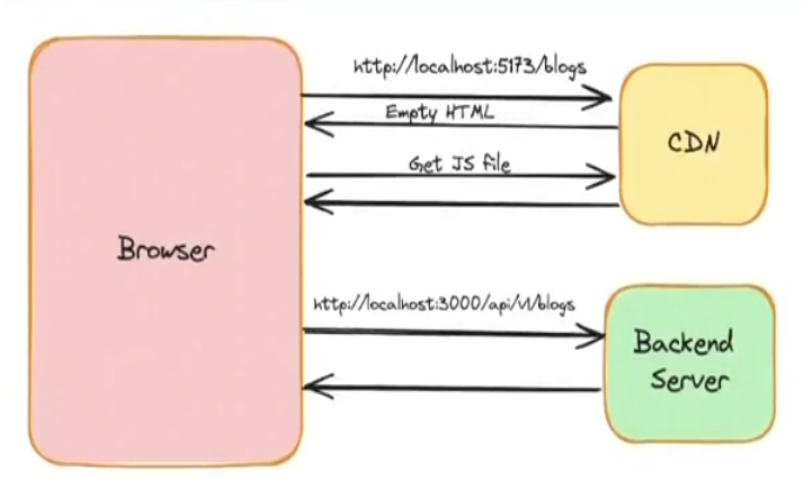
Delving deep into the above written code

```
export const UserCard = () => {
  const [userData, setUserData] = useState<User>();
  const [loading, setLoading] = useState(true);

  useEffect(() => { // Whenever the UserCard component mounts, we hit the backend
and get the data from the backend
    axios.get("https://week-13-
offline.kirattechnologies.workers.dev/api/v1/user/details")
      .then(response => {
        setUserData(response.data);
        setLoading(false);  // after getting the response, we set the loading to
false and the return the user data by using response.data
      });
  }, []);

  if (loading) {
    return <Spinner />; // basically loading screen dikhana agar data fetch nhi
hua h to
  }

    // and if data fetched then just show the below component
  return <div className="flex flex-col justify-center h-screen">
    <div className="flex justify-center">
      <div className="border p-8 rounded">
        <div>
          Name: {userData?.name}
        </div>
        {userData?.email}
      </div>
    </div>
  </div>;
}
```

Workflow of the above code

Although there are better ways to do the above thing using external library knwon as `ReactQuery`, but if you want to go purely with `REact`, then above is the only code to hit the `backend` from `frontend`.

Summarising the above concepts, **Data fetching happens on the client**



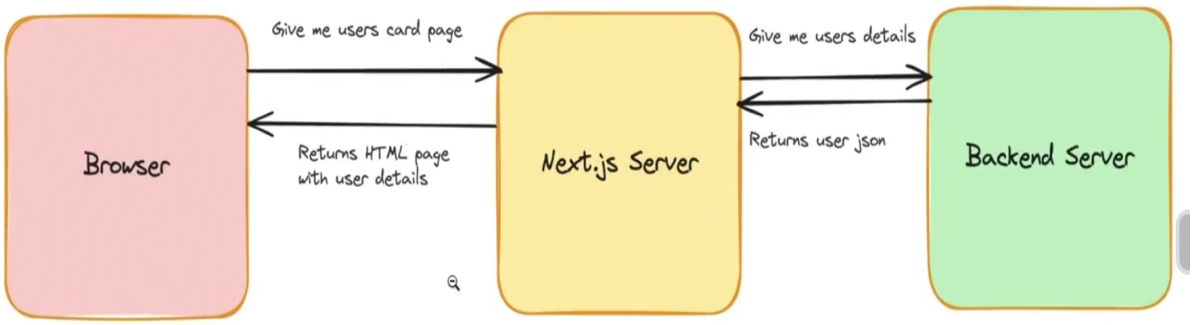
As discussed above, waterfalling problem exist while dealing with `React`.

Data fetching in `Next.js`

See the documentation -> Routing in Next.js

You should fetch the user details on the server side and **PRE-RENDER** the page before returning it to the user.

In `Next.js`, data fetching looks like the below :-



you can see the frontend is coming **PRE-RENDERED** from the `next.js` server.

Lets convert the above made `react` project to `next.js` project and build it

Step 1 -> Initialise an empty next project and do whatever option is required to create (you can also see in the previous page.)

```
npx create-next-app@latest
```

Step 2 -> for data fetching from the API(so that we can display that on frontend), install `axios` (you can also use `fetch`)

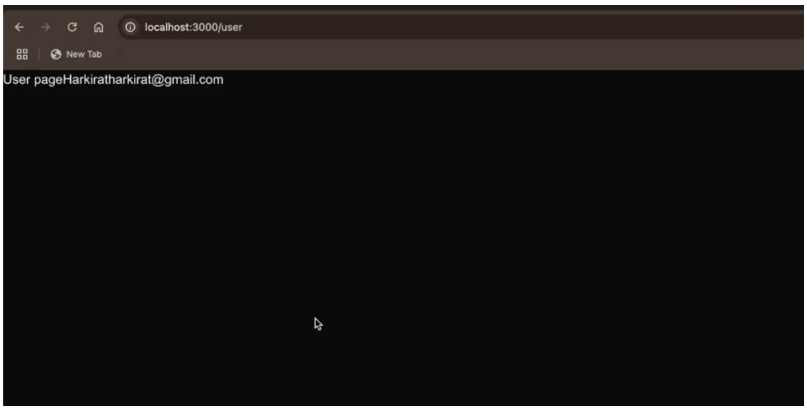
```
npm install axios
```

Step 3 -> Now go to the `app` folder and inside that make another folder(or route) `user` for showing the `user` data [which is our main goal]

Step 4 -> inside the `page.tsx` made inside the `user` folder writing the logic

```
"use client" // REMEMBER -> whenever you use "useState" or "useEffect" then you must add this line on top of your code when working with Next.js
// as these are CLIENT COMPONENT and by default Next.js SERVER COMPONENT maan ke chlta h sbko, so to use them we have to tell Next.js that they are CLIENT COMPONENT
import axios from "axios"
import {useEffect, useState} from "react"

export default function User(){
    // made 2 state variable as yhi done to screen pe show krna h first one (loading screen jb data nhi aaya ho to) and second one (data screen jb load ho jaye to)
    const [loading, setLoading] = useState(true)
    const [data, setData] = useState({})

    useEffect(() => {
        axios.get("https://week-13-offline.kirattechnologies.worker.dev/api/v1/user/details")
        .then(response => {
            setData(response.data)
            setLoading(false)
        })
    }, [])

    if(loading){
        return (
            <div>
                Loading....
            </div>
        )
    }

    return (
        <div>
            User Page
            {data.name}
            {data.email}
        </div>
    )
}
```

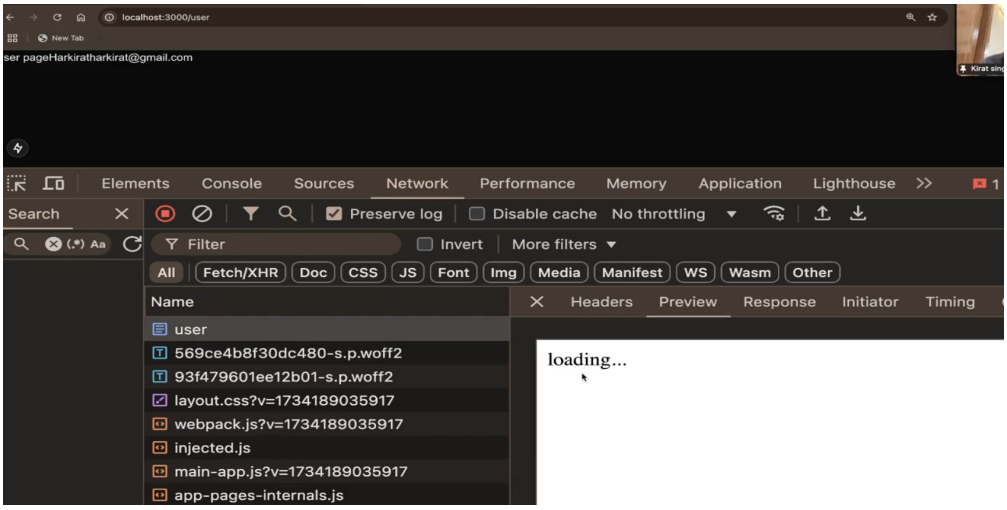
Output -> Initially loading will come and then the data.



BUT THE ABOVE THING IS NOT THE RIGHT WAY TO FETCH THE DATA INSIDE THE `Next.js`

Reason -> because the `useEffect` present inside the above code **will run on the `client` (means inside the BROWSER)**

simply saying, `useEffect` is running on the CLIENT side instead of SERVER and hence although waterfalling problem not exists here but still it is not getting the correct HTML page from the `next.js` server [Initial page looks like ->]



so `googlebot` will not see **first the data which is rendered on the screen instead it will see the loading page shown above and hence it will not be SEO optimised**

so although we use `next.js` still we were not able to take the benefit of it.

How to fix it ??

-> basically how to achieve `server-side-rendering` as currently it is doing `client-side-rendering`

Let do the changes inside the above code

```
import axios from "axios"

// STEP 1 -> make the component "async", so added async in the component
export default async function User(){ // 2
    // STEP 2 -> Remove all the logic of react hook written just do this
    const response = await axios.get("https://week-13-
offline.kirattechnologies.worker.dev/api/v1/user/details")
```

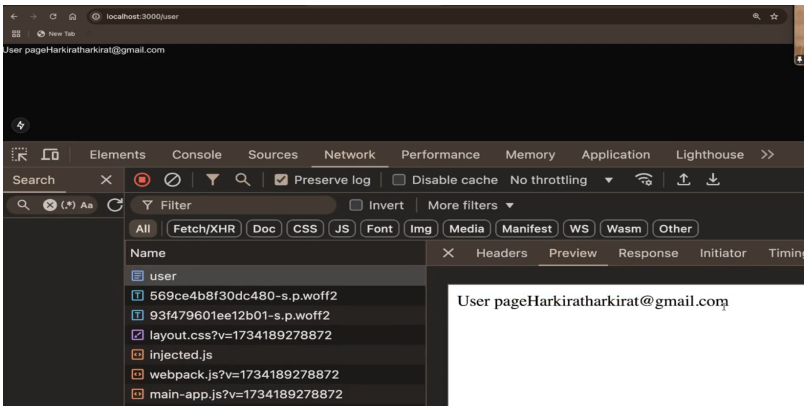
```
    const data = response.data // 3

    return ( // 4
        <div>
            User Page
            {data.name}
            {data.email}
        </div>
    )
}
```

Explanation of `// 2` code

writing `async` on the component does not means that *frontend pe `async` component is coming,* **it simply means that whatever logic written below till start of `return` codeblock WILL RUN ON THE SERVER, it will not run on the CLIENT**

basically `next.js` server ne `// 3` tk ka code run kiya and then `// 4 HTML` code ko **Pre-render krke bhej diya**

Now if you go and see the output ->



In the first `request` only, `HTML` page is returning the content which is rendered on the screen.**It is PRE-RENDERED on the server and getting loaded up (shuru se he bhara aa rha h backend se)**

Benefit of the above code ->

- **Code is much cleaner now**
- **`request` is directly going to the backend and page is pre-rendering and coming to the frontend**

> 📌**The above is how data fetching in `next.js` occurs, JUST REMEMBER THIS CODE as it is VERY VERY IMPORTANT**

Basically your `next.js` is either INSIDE the backend server or your `next.js` server is talking to the database

The above was one of the biggest thing `next.js` took an edge over `react` and that is USE OF STATE MANAGEMENT LIBRARY like `recoil` totally came to end, as jarurat he nhi pdta to store the data inside the state variable and then render them, sb bana banaya aata h

Now the question comes,

💡 **Lets say in the above code, the data took 2 or more second to load then how to render the LOADER till the data is loaded as we have not written or done anything about that ??**

Loaders in `Next.js`

---

so here comes the **requirement of your website ->** if you want your website to be SEO optimised, then **loading page to dikha mt dena as if you will add that then the first `response` coming from the server will be LOADING page only as isse jldi to data load hoga he and due to this `googlebot` will see the LOADING page which will eventually drastically reduce the SEO optimisation**

Now here `googlebot` will never go to the each users (i.e. go to the `user` route), to know about each user, so here its **better to add LOADER** as `googlebot` ko ye sb route se mtlb utna nhi h

BUT

if you are making your landing page of the website then you cant add LOADER page as then it will NOT BE SEO optimised as the website function is mostly shown inside the landing page only and `googlebot` mostly reads this only.

so if you want to add LOADER then go to the `react` approach of adding the LOADER page written above

OR

A more `next.js` approach is ->

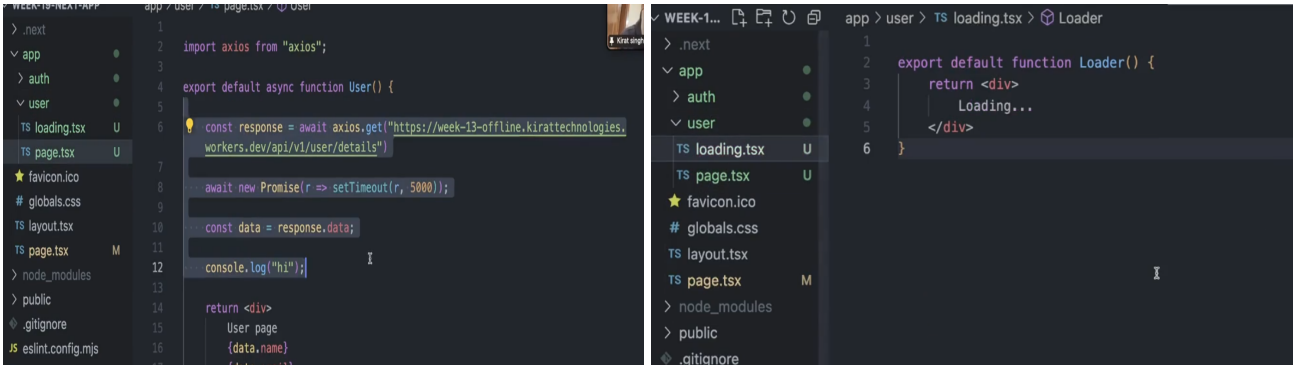
```
import axios from "axios"

export default async function User(){
    const response = await axios.get("https://week-13-
offline.kirattechnologies.worker.dev/api/v1/user/details")

    const data = response.data

    return (
        <div>
            User Page
            {data.name}
            {data.email}
        </div>
    )
}
```

If you know that there are bunch of `async` task is being happening inside the compenent, then add another file (or page) called as `loading.tsx` inside the same folder as that where you have written the above code i.e. -> `app > user > page.tsx`

and inside `loading.tsx`

```
export default function Loading(){
    return (
        <div>
            Loading... // add what ever the loading page you want to add here
        </div>
    )
}
```

see the folder structure :-



Now the `next.js` is clever enough ki **jb tk left pic me highlighted part (`async` code) is fetching the data and is taking time, tb tk as `loading.tsx`(right pic) v usi route me h so run its code (WHY TO SIT EMPTY and wait for the data) and hence it will show LOADER by executing the `loading.tsx` file and you will see the LOADER till the data is being fetched or simply saying till `page.tsx` ki `async` code nhi chal jati**

> 📌**Basically FALLBACK to `loading.tsx` page (if present) if one page is taking too much time to load up. [CONDITION -> all the pages including `loading.tsx` should be in same folder]**
>
> > **REMEMBER -> Dont change the name of `loading.tsx` (ye `loading.tsx` he hona chahiye)**

Initial `HTML` me isse v `loading.tsx` ka `HTML` code he aayega but now you have done it in `next.js` way instead of doing it like `react` way

Introducing API routes in `Next.js`

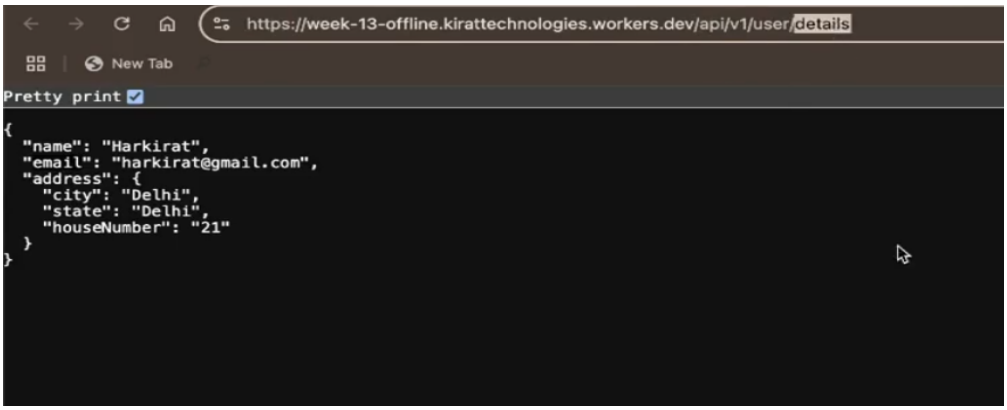
NextJS lets you write backend routes, just like express does.

This is why Next is considered to be a `full stack` framework.

The benefits of using NextJS for backend includes

1. Code in a single repo
2. All standard things you get in a backend framework like exprbss
3. Server components can directly talk to the backend

Till now we were writing the `page handler`, now we will write `route handler`



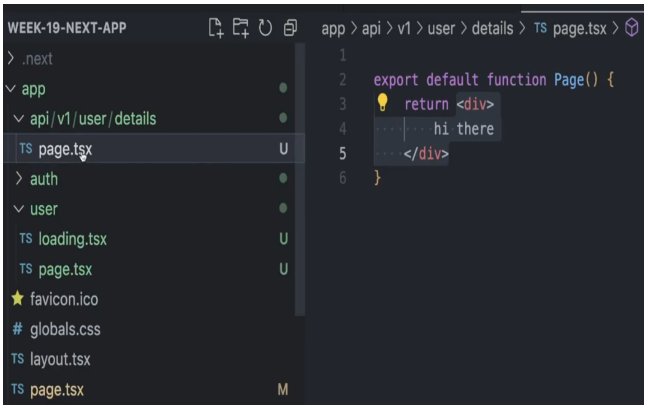
See the url, as this url was externally present means it was not inside the `next.js` server means `next.js` server has fetched the data from external backend. Now to make it available inside my `next.js` server, i will create same route ->

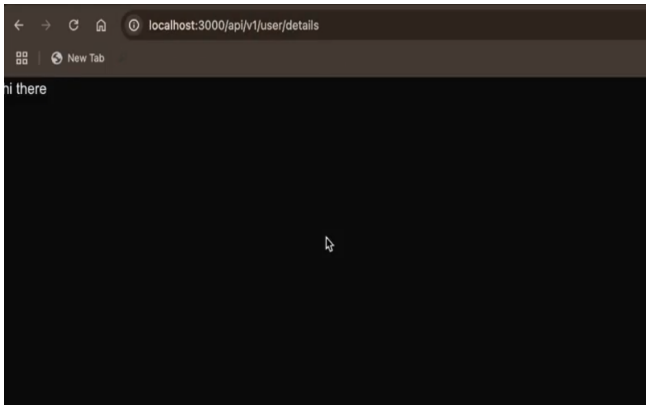
so going inside the `app` folder, adding now `route handler` and then inside that make first `api` folder, then inside that `v1` folder, and then inside that `user` folder, and finally inside that `details` folder inside which if you make `page.tsx` file inside which if you write :-

```
export default function Page(){
  return (
    <div>
      hi there
    </div>
  )
}
```

Now if you now go to "http://localhost:3000/api/v1/user/details"

then you will see "hi there" on the screen but what do i want to see at this -> see the above pic present in `json` format





so can i write like this :-

```
export default function Page(){
  return {
    name : "harkirat"
```

```
  }
}
```

No this will give ERROR if you see it on the `url` as it is component and it EXPECTS `JSX` to return

Now as **I am not returning a page(not returning a `HTML` from here), IT IS A BACKEND route(it might return `text`, `json`, and maybe `HTML`)** so for that

Convert the name of the file to `route.tsx` (doesnt even have to `.tsx` as we are currently not returning `HTML` (see the data it is in `JSON` format)) so `.ts` is enough

as you have already see the frontend part in `next.js`, recaping the steps :-

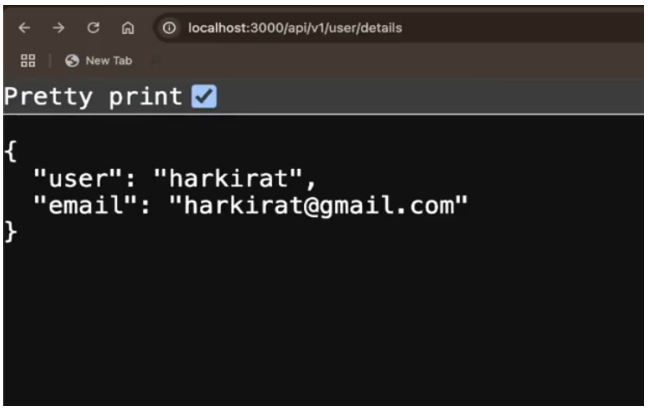
For pages , return a component that return some `HTML`, above which data fetching is code is being handled

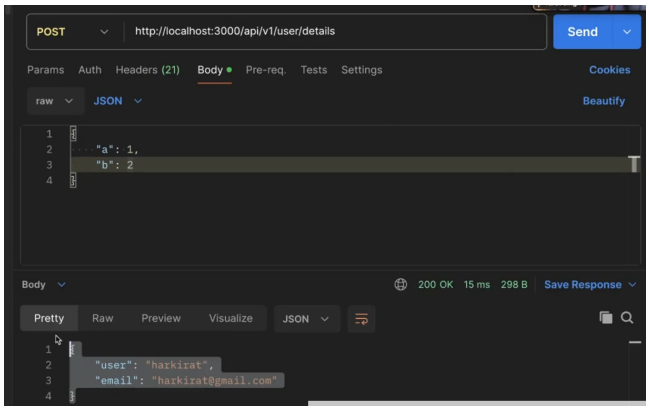
Now coming to the `backend` part,

now inside the `route.ts`, write this -> [Here we will learn the syntax for backend]

```
import {NextResponse} from "next/server" // first importing the NextResponse from
the next/server

// THIS IS HOW YOU MAKE "GET" REQUEST
// you have made a "get" request on the
// as we have MULTIPLE type of handler on the route -> api/v1/user/details so that
why not used DEFAULT on the function (see the Default vs Normal export difference)
export function GET(){
  return NextResponse.json({
    user : "harkirat",
    email : "harkirat@gmail.com"
  })
}

// THIS IS HOW YOU WILL HANDLE POST REQUEST
export function POST(){
  return NextResponse.json({
    user : "harkirat",
    email : "harkirat@gmail.com"
  })
}

// SIMILARLY HANDLING THE PUT REQUEST
export function POST(){
  return NextResponse.json({
    user : "harkirat",
    email : "harkirat@gmail.com"
  })
}

// IN THE SAME WAY YOU CAN HANDLE ANY TYPE OF REQUEST
```

Output -> `get` request (in left side) and `post` request (in the right side)



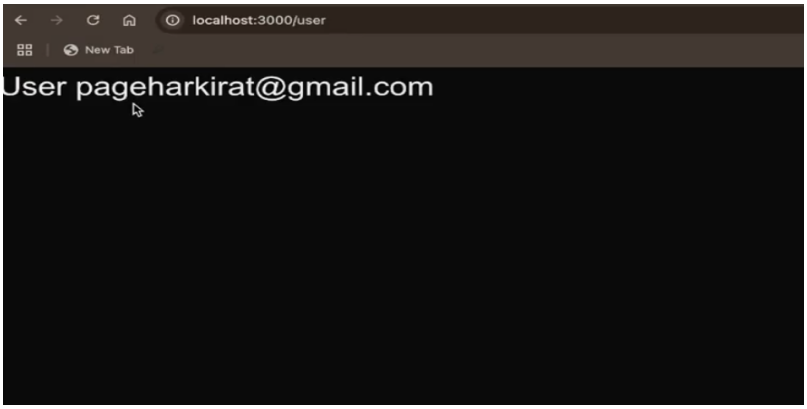


> 📌 **REMEMBER this line ->`NextResponse` in `next.js` is same as `res` used in `express`**

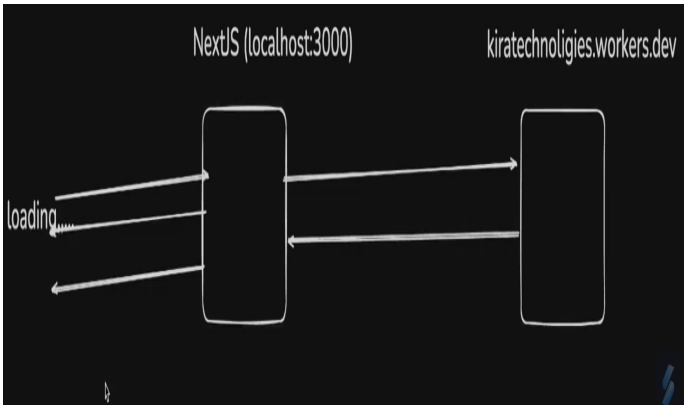
Now you can use this in your intial code where you were hitting other backend

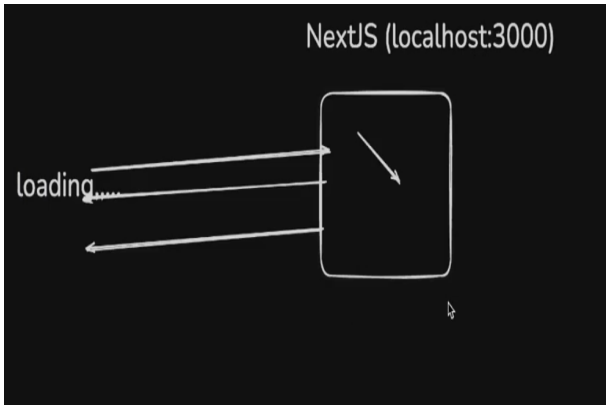
```
import axios from "axios"

export default async function User(){
    // const response = await axios.get("https://week-13-
offline.kirattechnologies.worker.dev/api/v1/user/details") // INSTEAD OF THIS

    // WRITE THIS
    const response = await axios.get("http://localhost:3000/api/v1/user/details")

    const data = response.data

    return (
        <div>
            User Page
            {data.email}
        </div>
    )
}
```

Output ->



what we were doing initially -> (in the left pic) and what we are doing now (in the right pic, basically Nextjs pe he bhej rhe ho request)





so we have MIGRATED the backend to our `next.js`

Project -> End to End To-do application in `next.js`

Initialising an empty `next.js` project by the same step done previously ->

and then proceeding by `page.tsx` present inside the `app` folder ->

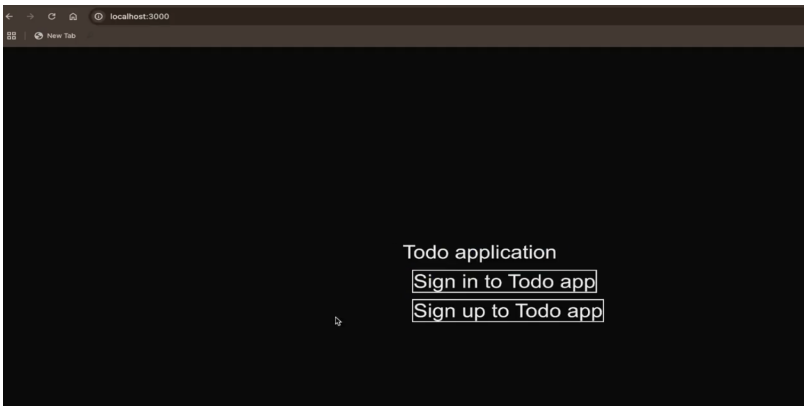
inside the `page.tsx`

```
import Link from "next/Link"

export default function Home(){
  return (
    <div className = "text-lg w-screen h-screen flex items-center justify-center">
      <div>
        TO-DO application
        </br >
        <Link className = "text-md border m-2" href = "/signin">Sign in to TODO
app</Link>
        </br >
        <Link className = "text-md border m-2" href = "/singup">Sign up to TODO
app</Link>
      </div>
    </div>
  )
}
```

> 📌 **`<Link>` tag is present in `next.js` and works the same way as `a` tag in `HTML`**
>
> > **Used to RE-DIRECT to other page (also it is a button, you click on it)**

> 📌 **REMEMBER -> Using `<Link>` tag is the BEST way to do ROUTING in `next.js` although there exists another way but that will lead to complication only**

Going to the http://localhost:3000 , you see the output as ->



If you click on the sign in then it will redirect to the http://localhost:3000/signin ,(i.e -> `signin` page) and similar with the `signup` page

Other way of doing routing inside `next.js`[Complicated way]

```
"use client"
import {useRouter} from "next/navigation"

export default function Home(){
  const Router = useRouter()
  return (
    <div className = "text-lg w-screen h-screen flex items-center justify-center">
      <div>
        TO-DO application

        <button onCLick = {() => {
          Router.push("/sigin")
        }}>Sign in</button>
      </div>
    </div>
  )
}
```

> 📌 **using `useRouter` is another way to achieve ROUTING in `next.js` but this BAD way to do as you have to make the code CLIENT SIDE as it is available in CLIENT SIDE which will enforce the code to run on the CLIENT SIDE and `next.js` is known for SERVER side** also **PRE-FETCHING is not available for this way of doing routing as CLIENT SIDE me possible nhi hota, go above, we have discussed it**
>
> > **in the code at the top i have added line `"use client"` as it is CLIENT SIDE component**

The code will still work but you should avoid using this way of doing routing.

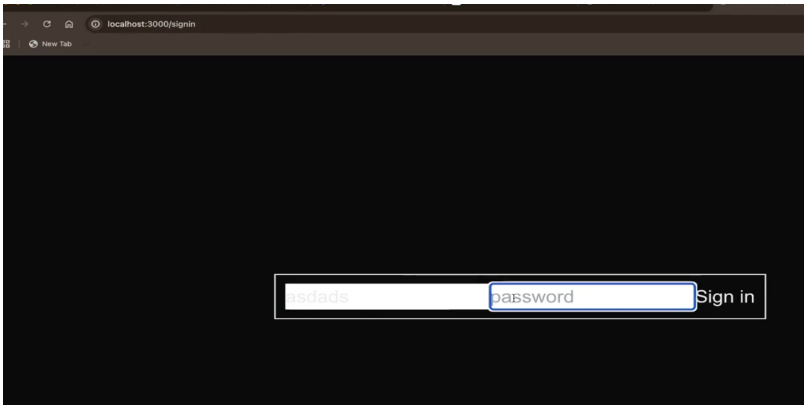
Now next step is -> Creating the `signin` and `signup` page so inside the `app`, make a folder named as `signin` and inside that `page.tsx` to write the logic for this page, the code for it ->

```
"use client"
import axios from "axios"
```

```
export default function Signin() {
  return (
    <div className="w-screen h-screen flex justify-center items-center">
      <div className="border p-2">
        <input type="text" placeholder="username"></input>
        <input type="password" placeholder="password"></input>
        <button onClick={() => {
          axios.post("http://localhost:3000/api/v1/signin") // we have to create
this endpoint and make it hit the database so that data can be fetched and for
that we will take username and password given by the user above in the field made
        }}>
          Sign in
        </button>
      </div>
    </div>
  );
}
```

> 📌 **Whenver you add `<button>` component in `next.js`, as it is CLIENT SIDE COMPONENT so you have to add `"use client"` at the top of your code**

Output ->



similarly make a folder named as `signup` inside the `app` folder and inside that `page.tsx` to write the logic for this page, the code for it ->

```
"use client"
import axios from "axios"

export default function Signup() {
  return (
    <div className="w-screen h-screen flex justify-center items-center">
      <div className="border p-2">
        <input type="text" placeholder="username"></input>
        <input type="password" placeholder="password"></input>
        <button onClick={() => {
          axios.post("http://localhost:3000/api/v1/signup", { // same we require
to make this endpoint and hit the database but for that first we need to take the
username and password given in the input field // How to do that ?? // 2
```

```
          })
        }}>
          Sign up
        </button>
      </div>
    </div>
  );
}
```

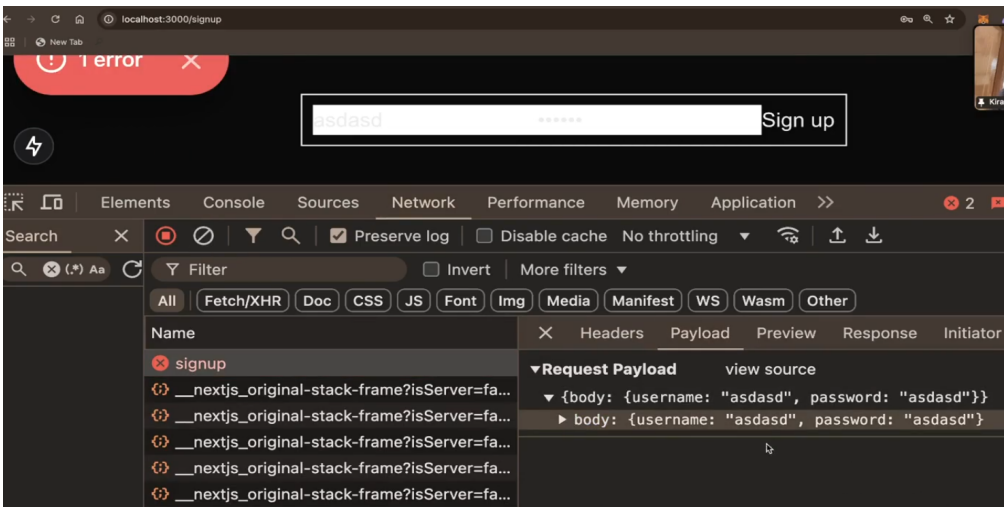
Answer to `// 2` question

There are two ways to do the above thing ->

1st way -> using STATE Variables

```
"use client"

import axios from "axios"
import {useState} from "react"

export default function Signup() {
  const [username, setUsername] = useState("") // made two variable that if any
change in their value occurs then it should re-render the component
  const [password, setPassword] = useState("")
  return (
    <div className="w-screen h-screen flex justify-center items-center">
      <div className="border p-2">
        <input type="text" placeholder="username" onChange = {e => // added the
onChange handler that if any change happens in the username then an event should
occur, e.target -> as the event ispe hua h to e.target se isko refer kiya jaa rha
h and e.target.value -> means finally input box ke andar jo value h usko refer
kiya jaa rha h
          setUsername(e.target.value)
        }></input>
        <input type="password" placeholder="password" onChange = {e =>
          setPassword(e.target.value) // same explanation as above
        }></input>
        <button onClick={() => {
          axios.post("http://localhost:3000/api/v1/signup", {  // sending the
username and password
            username,
            password
          })
        }}>
          Sign up
        </button>
      </div>
    </div>
  );
}
```

Output ->



You can see that data is getting in the correct way to the backend (although we still have to implment the backend)

2nd Way -> using `useRef` hook (as you have expected), we have already learnt that this can be used to REFER and THIS WAY IS BETTER WAY as RE-RENDER will NOT happen in this way every time the username or password changes

💡 How to implement backend part

-> already done above so using the same concept to achieve this

create a folder `api` inside the `app` folder, then inside that `v1`, then inside it `signup` and then inside it create a file named as `route.ts` (where the code looks like this)

```
import {NextRequest, NextResponse} from "next/server"

export async function POST(req : NextRequest){ // How to get request (req) here for this here you GET ACCESS TO "req" OBJECT whose type is NextRequest and using this object you can EXTRACT whatever possible while dealing with POST request (ex -> query parameters, headers, body etc..)

  // BELOW IS THE WAY TO EXTRACT THE BODY
  const data = await req.json() // this will extract the BODY coming from the frontend (which in our case is username and password)
  // Now you have got the data in the backend next step will be storing in the database HOW TO DO THAT (see below)


  return NextResponse.json({ // we know that NextResponse act same as res in express and used to send the response as res does in express but what about getting the data means "req" in express for this NextRequest also exists
    message : "you have been signed up"
  })
}
```

> 📌 **just REMEMBER -> `NextRequest` in `next.js` is same as `req` in `express` and `NextResponse` in `next.js` is same as `res` in `express`**
>
> > **As you are using them inside the `Next.js` so added "Next" in front of it**

💡 **How to redirect to `signin` page after the user has done `signup` ??**

-> inside the `signup`'s `page.tsx` add these ->

```
"use client"
import {useRouter} from "next/navigation"
import {useState} from "react"
import axios from "axios"

export default function Signup() {
  const [username, setUsername] = useState("")
  const [password, setPassword] = useState("")
  const router = useRouter() // used router as we want to redirect
  return (
    <div className="w-screen h-screen flex justify-center items-center">
      <div className="border p-2">
        <input type="text" placeholder="username" onChange = {e =>
          setUsername(e.target.value)
        }></input>
        <input type="password" placeholder="password" onChange = {e =>
          setPassword(e.target.value)
        }></input>
        <button onClick={async () => { // make this function "async" as it will
take some time to go to backend and from there storing the username and password
to the database
          axios.post("http://localhost:3000/api/v1/signup", {
            username,
            password
          })
          router.push("/signin") // once data is stored in the database, redirect
this to "signin" page. You could have also used Link tag for routing as discussed
above but as you have see its implementation above so using this way now
        }}>
          Sign up
        </button>
      </div>
    </div>
  );
}
```

Now if you go to the `signup` page and then put some data and then click on `sign up` button then it will now REDIRECT to the `signin` page as given in the question above.

Storing the data in the database

Now that you have got the data in the backend, your next step will be **STORING IT IN THE DATABASE**

so to do that again coming to the code we have written till now in our backend created inside the `next.js`

```
import {NextRequest, NextResponse} from "next/server"


export async function POST(req : NextRequest){

  const data = await req.json()
  // you have to do somthing like the below
  mongoose.User.insert() // if using monogoose
  pg.query("INSERT INTO USERS sql query ahead") // if using postgres
  prisma.user.update({data to be sent}) // if using prisma

  return NextResponse.json({
    message : "you have been signed up"
  })
}
```

Here we will storing in the database through `prisma`, so using it to store inside the database :-

Adding `prisma` to `next.js` project

---

for using `prisma` using `postgres`, you already know how to setup that

Step 1 ->get the `postgres` sql connection url from the website -> neon.tech

Step 2 -> Install prisma

```
npm install prisma
```

Step 3 -> Initialise `prisma` schema

```
npx prisma init
```

-> The above command created `schema.prisma` file and inside which we have to write the logic

```
generator client {
  provider = "prisma-client-js"
}

datasource db{
  provider = "postgresql"
  url = env("Database_URL")
```

```
}

Model User {
  id    Int   @default(autoincrement())   @id
  username     String     @unique
  password     String
}
```

after this, make sure to ADD the Dastbase url got from the `neon.tech` inside the global `.env` file.

Step 4 -> Migrate the database(`schema.prisma` file ka jo table h, usko database me REFLECT to krwana pdega)

```
npx prisma migrate dev --name init_schema
```

Step 5 -> Generate the client

```
npx prisma generate
```

-> this will create folder to ./node_modules/@prisma/client [**These folders have the file that can be used to connent to the database**] for ex -> Object oriented programming ki takat de dega ye

```
import {NextRequest, NextResponse} from "next/server"
import {PrismaClient} from "@prisma/client/extension" // Import the client
generated

const prismaClient = new PrismaClient() // making a variable which has the
capability of it
export async function POST(req : NextRequest){

  const data = await req.json()

  // TO FURTHER ENHNACE, ADD ZOD HERE FOR VALIDATION OF THE INPUT

  await prismaClient.user.create({ // then using the prismaClient to gain the
capability for interacting with the database
    data : {
      username : data.username,
      password : data.password
    }
  })

  return NextResponse.json({
    message : "you have been signed up"
  })
}
```

Now if you go the `signup` page, put some entry for `username` and `password` and then click on `signup` button, you will be redirected to `signin` page and at the same time, **Data you put while on the `signup` page will be stored inside the `postgres` database**, just REFRESH the made database made on the `neon.tech` site and you will see that inside the `User` table, you will see the **new entry of the `username` and `password` you have given while on the `signup` page**

Better fetches

Now lets create an Endpoint which **will return you the details making it**

For the root page, we are fetching the details of the user by hitting an HTTP endpoint in

```
import axios from "axios";

async function getUserDetails() {
  try {
    const response = await axios.get("http://localhost:3000/api/v1/user"); // 2
    return response.data;
  } catch (e) {
    console.log(e);
  }
}

export default async function Home() {
  const userData = await getUserDetails();

  return (
    <div className="flex flex-col justify-center h-screen">
      <div className="flex justify-center">
        <div className="border p-8 rounded">
          <div>
            Name: {userData?.name}
          </div>
          {userData?.email}
        </div>
      </div>
    </div>
  );
}
```

and then handling `GET` request present on the Home page (**Basically we are trying to give all the data associated with the user on the Home screen**)

```
import {NextRequest, NextResponse} from "next/server"
import {PrismaClient} from "@prisma/client/extension"

const prismaClient = new PrismaClient()
export async function POST(req : NextRequest){
```

```
  // SAME AS ABOVE CODE
}

// ADDING THE GET REQUEST here
export async function GET(req : NextRequest){ // 3
  const user = await prismaClient.user.findFirst();

  return NextResponse.json({
    user
  })
}
```

You are basically **fetching the data from the backend server (which in turn is fetching from the database) and then finally you are showing the corresponding data of the user on the Home page**

The above code is working like this -> you are hitting your own backend (see `// 2`) line of code and when this line of code executes, you **hit your own made backend** and then **the control reaches at `// 3` line of code and hence you get the user details**

But there can be BETTER way to FETCH ->

Basically **why you have to hit a seperate api(basically hit your own backend)**

Why not write the `get` logic inside the component only ??

something like this :-

```
import axios from "axios";

async function getUserDetails() {
  try {
    // const response = await axios.get("http://localhost:3000/api/v1/user"); //
INSTEAD OF HITTING YOUR OWN BACKEND and then run the logic from that
    // return response.data; // DIRECTLY WRITE THIS LOGIC HERE ONLY
    const user = await client.user.findFirst({}) // 2
    return ({
      name : user?.username,
      email : user?.email
    })

  } catch (e) {
    console.log(e);
  }
}

export default async function Home() {
  const userData = await getUserDetails();

  return (
    <div className="flex flex-col justify-center h-screen">
      <div className="flex justify-center">
        <div className="border p-8 rounded">
```

```
            <div>
              Name: {userData?.name}
            </div>
            {userData?.email}
          </div>
        </div>
      </div>
    );
  }
```

Basically, **You dont really need to hit your own made `HTTP` server, you can directly do this in the component only**

SCARY PART of the above apporach -> **Mere component me database he aayega** and then the question arises

💡 **can this `client` be hit from the frontend (as it has came in the frontend to tm database (i.e. `client`) ko power de diye hit krna ka iska mtlb), WILL THE USER CAN HIT THE DATABASE ??**

-> **Answer to the above question is NO (a big NO) because on the client side or basically user side, final `HTML` code is only returned NOT THE LOGIC so whatever the `async` function you are doing here will REMAIN ONLY ON THE SERVER and run on the server not the client and hence the user cant hit the database from frontend.**

> 📌**Remember only the `HTML` part is returned to the `user` not the logic for backend if any in `next.js`**

So rather that creating than hitting yourself on the different API endpoint, dont create the API endpoint instead DIRECTLY DO THE DATABASE SEARCH OVER THERE.

in short, **if you just want to do SEARCH, then DO THE SEARCH DIRECTLY**

Singleton Prisma client

Click to see the documentation -> Prisma client Troubleshoot

The above and below thing is the `next.js` specific topic

You will not face this issue when you are in PRODUCTION, (if you deploy your website, it will totally be fine), also you will not face this issue when you are running this locally but using the local database BUT if you are running the code locally but your database is not local (This is our case or the way we have coded above or made the website) like our database is on `neon.tech` so in this case, if are doing **HOT RELOADING (means you are making changes in your codebase too frequently and then running the code, this will create TOO MUCH CONNECTION(as `const prismaClient = new PrismaClient()` line will RUN every time) `request` to the database which will GIVE YOU AN ISSUE that you have too many connection to the database** [This HOT RELOADING thing does not comes when you deploy it as there, code is once submitted you do not make frequent change in the database].

To fix that -> we **create a singleton**

💡 What is Singleton ??

-> Singleton basically means that **"Something which is defined only once"** (doesn't have multiple references)

the above is what we exactly want

Whenever the code is compiling, the line `const prismaClient = new PrismaClient()` **should be restricted to one time running only**

for that **Put this line of the code to the seperate folder and name it as `lib` (not matter) which is present inside the `app` folder and inside that make `db.ts` file whose content are as follows**

```
import {PrismaClient} from '@prisma/client'

const prismaClientSingleton = () => { // 2
  return new PrismaClient()
}

// @ts-ignore  (just wrote to ignore any ts error if being given by the ts)
const prisma = globalThis.prisma ?? prismaClientSingleton() // 3

export default prisma

if(process.env.NODE_ENV !== 'production') globalThis.prisma = prisma // 4
```

Explanation of `// 2` code

What this line `const prismaClient = new PrismaClient()` is doing the same thing is done by the `// 2` codeblock.

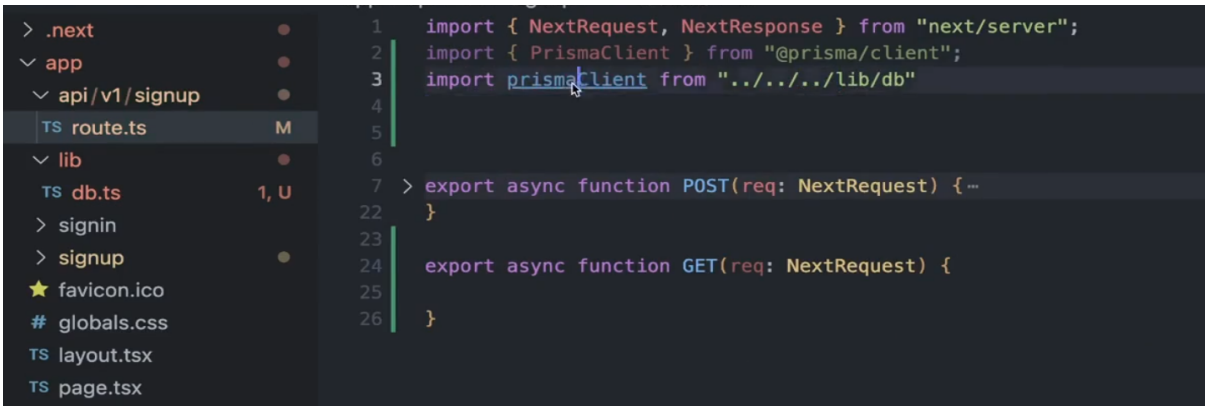
Explanation of `// 3` code

This line means that either `prisma` is `globalThis.prisma` (**`globalThis` means simply `WINDOW` object which is present inside the browser (acts as GLOBAL OBJECT))**)

Explanation of `// 4` code

This line means that if the **codebase is not running into the production, then do make that GLOBAL variable = `prisma`**

Basically now you will do something like this ->



Notice you are importing the `prismaClient` from the `db.ts` present inside the `lib` folder, **Now as the line (`const prismaClient = new PrismaClient()`) is not present here so if reloading will happen this will not trigger to again connect to the database**

Now First time the code will run `globalThis`, as it is NULL its value will become equal to `new prismaClient`, and then next time it will run, as `prismaClient` is already being made so this code will not re-run hence avoid **RE-CONNECTING to the database frequently**[TOO MANY CREATION OF `prismaClient` WILL BE AVOIDED]

How to do it via the `mongoose` ??

-> same process make a folder named as `lib` inside the `app` and inside which create the file named as `db.ts` and inside that

Step 1 -> First install the `mongoose` as dependncy by running the command

```
npm install @types/mongoose
```

Step 2 -> Define the schema as you define while you were working with `mongoose`.

```
import mongoose, {Schema, Model} from "mongoose"

mongoose.connect("Database_URL") // put the url inside the .env file

const userSchema = new Schema ({
  username : String,
  password : String
})

export const UserModel = Model("user", userSchema)
```

Now making changes according to the above in the `api/v1/signup` and then inside that present `route.ts`

```
import {NextRequest, NextResponse} from "next/server"
import {PrismaClient} from "@prisma/client/extension"
import {UserModel} from "@lib/db"
```

```
const prismaClient = new PrismaClient()
export async function POST(req : NextRequest){
   const data = await req.json()

  // TO FURTHER ENHNACE, ADD ZOD HERE FOR VALIDATION OF THE INPUT

  // await prismaClient.user.create({ // then using the prismaClient to gain the
capability for interacting with the database
  //   data : {
  //     username : data.username,
  //     password : data.password
  //   }
  // })
  // INSTEAD OF WRITING THE ABOVE CODE as we are now using "mongoose" so add
"mongoose" related code to insert into the mongoDB as this is the database we are
using
  UserModel.insert()

  return NextResponse.json({
    message : "you have been signed up"
  })
}


export async function GET(req : NextRequest){
  // SAME AS CODE WRITTEN ABOVE
}
```

Some important Questions realted to `next.js`

💡 What is HOT MODULE RELOADING in `next.js` ??

Without the page reloading, the changes which you will make or occur in the page gets reflected on the page, this is what is called as HOT MODULE RELOADING means in `next.js`

> 📌 **It simply meaans MODULE TO RELOAD HO RHA H BUT PAGE RELOAD NHI HOTA H**

💡 What is caching ??

-> You **store the important thing inside the browser** so that you dont have to hit the database frequently to get or verify the data

💡 What does SERVERLESS means ??

-> SERVERLESS does not means that without server your codebase is running, it simply means that you have given your code to someone who will deploy your code on the server (which you are not aware of) and that has all the responsiblity to handle your site, and in turns charge you per request for handling that

This is different from `cloud` here you know where your codebase has been deployed and through the `cloud service provider` you are paying them and in turn they are responsible for maintaining your codebase[simply means you have rented a machine via the `cloud service provider` through which your codebase is running]